Dokumentation

**Inhalt**

[**Vorwort** 2](#_Toc139004206)

[**Spielprinzip** 3](#_Toc139004207)

[**Vorbereitung** 3](#_Toc139004208)

[**Erster Plan** 3](#_Toc139004209)

[**Kurzfassung des Verlaufs** 4](#_Toc139004210)

[**Bibliotheken** 5](#_Toc139004211)

[**Programm** 5](#_Toc139004212)

[**Klasse** 5](#_Toc139004213)

[**Ressourcen** 9](#_Toc139004214)

[**Sprites** 9](#_Toc139004215)

[**Sound** 9](#_Toc139004216)

[**Fazit** 9](#_Toc139004217)

[**Quellenangaben** 9](#_Toc139004218)

# **Vorwort**

Ein Spiel programmieren haben wir gedacht, es wäre einfach haben wir gedacht. Kurzgefasst, wir haben uns etwas Zuviel vorgenommen. Eine Mischung aus zu hohen Erwartungen, großen Ambitionen und etwas zuviel Kreativität bei der Planung haben uns gelehrt was es heißt, etwas selbst zu programmieren.

Dennoch wagen wir es zu behaupten, wir haben viel gelernt. Nicht nur, zu verstehen, dass der ein oder andere Selbstanspruch ungesund sein kann, sondern auch viel in der Programmierung selbst. So viel, wie in den letzten 1 ½ Monaten hatten wir in Java noch nie gelernt. Das ist unsere Odyssey vom ersten Planen   
bis zum (un)vollendeten Spiel.

# **Spielprinzip**

Die Idee war es ein Spiel zu programmieren das dem Rogue-Like Genre zugeordnet werden konnte. Inspiriert von spielen wie "The Binding of Isaac" und Dungeon Crawlers war es das Ziel ein eigenes, ähnliches Spiel zu entwickeln. Der Spieler sollte sich durch mehrere Räume durchkämpfen bis zu ein Endboss. Ein genaues Thema, Aussehen etc. wurde dabei noch nicht vor der Planung aufgestellt, lediglich etwas Brainstorming. Ideen wie schießbare Projektile, unterschiedliche Mobs, Items und NPCs zur Erweiterung des Spielumfangs und oder der Story waren im Gespräch. Wie sich recht schnell herausstellte war das alles für die Zeit und unsere Kenntnisse viel zu weit gegriffen. Somit unterscheidet sich das fertige Produkt in vielen Hinsichten der eigentlichen Idee.

Das Produkt wurde Weit nicht so umfassend wie Erwartet. Beim Starten des Spiel gelangen wir in den Startbildschirm, der einen Hintergrund und 3 Verschieden Buttons zu besitzt. Vor dem Spielen, kann man in den Einstellungen die Jeweilige Sprache einstellen oder die Credits durchlesen. Sobald man das Spiel startet landet man in einem 32 mal 32 Blöcke großen Raum, in dem man sich frei bewegen kann mit den Tasten WASD. Mit den Mauseingaben linksklick und rechtsklick, kann man mit dem Schwert schlagen beziehungsweiße mit dem Schild blocken. Dafür gibt es nur eine Konsolenausgabe, aber kein Animationen. Außerdem wird erkannt ob sich ein Mob in einem 3 mal 3 großen Radius befindet oder nicht. Der Spieler kann als auch der Mob können das Spielfeld nicht verlassen und wird durch die grauen Blöcke abgegrenzt. Der Spieler kann in dem Feld Münzen einsammeln, die Anzahl der Münzen wird dabei oben rechts genannt. Ebenfalls läuft im Hintergrund das Lied „Tanz auf dem Vulkan“, was von Marc selbst erstellt wurde.

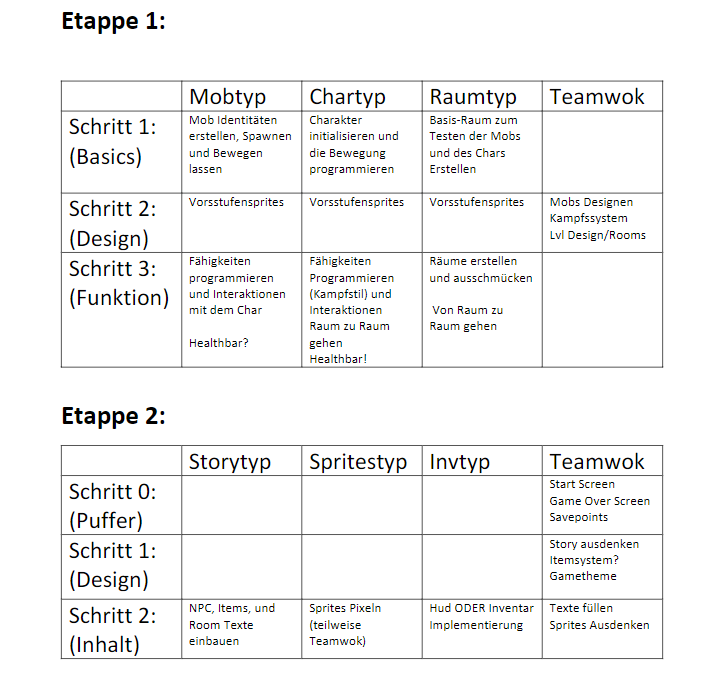
# **Vorbereitung**

Unsere Vorbereitungsphase begann direkt mit einer Überraschung. Durch das spontane Ausfallen eines Mitschülers betritt Jason Hein als drittes Mitglied des Projektes am Tag der eigentlichen Vorplanung. Daraufhin stellten wir folgenden Plan zusammen:

## **Erster Plan**

Nachdem wir uns eine erste grobe Idee erarbeiteten für unser Rogue-Like-Spiel teilten wir den Plan in ein Etappenprinzip ein.

**Jason**s Aufgabe (Mobtyp) war es sich um die Gegner zu kümmern, diese zu designen, implementieren und beweglich zu machen.   
**Marc**s Aufgabe (Chartyp) wiederrum war sich an den Spieler selbst zu wenden, diesen beweglich machen zu können und Fähigkeiten auszuführen  
**Leon** baute währenddessen das Spiel-interface und den Raum des Spiels selbst

**Erklärung:**  
Das eigentliche Ziel war es Etappe 1 abzuschließen. Diese wurde in 3 Schritten aufgeteilt. Schritt 1 stellte die Basics dar. Jeder sollte die ersten Objekte seiner Aufgabe erstellen sodass wenn Schritt 2 erreicht wurde, man durch simpelsten Sprites die Bewegungen auf dem Spielfeld sehen konnte. Schritt 3 würde dem Spiel dann letztendlich seine Funktion geben.

Etappe 2 & die derzeit undefinierte Etappe 3 wären zusätzliche Inhalte, doch wie so immer kommt es anders...

# **Kurzfassung des Verlaufs**

Projektzeitraum: 15.5. - 30.6.2023

Nach der Aufstellung unseres ersten Plans trafen wir uns am 15.5. persönlich. Doch die ersten paar Stunden standen nicht unter einem guten Stern. Wir versuchten letztendlich vergebens JavaFX funktionabel für uns alle zum Laufen zu bekommen doch nach einigen Gesprächen stellten wir fest dass wir eine andere Lösung brauchten. Bessere Lösung ist bei Java.Swing leider nicht der Begriff der Wahl, aber der Vorteil ist eindeutig. Es funktioniert, und das konstant. Somit begann die Reise in den Codewahnsinn und wir fingen mit den ersten Zeilen an

Am Anfang arbeiteten wir eine kleine Klassenübersicht aus. Gemäß der Einteilung gab es die Klassen für die GUIs, SpielPanel, StartBildschirm und SpielFrame die hauptsächlich von Leon angegangen wurde. Recht schnell stand schon der kleine Startscreen, den man aus dem Programm kennt. Jason erstellte die Mob Klasse und implementierte ein KeyListener für die Steuerung. Marc kümmerte sich um die Charakter Klasse und dem Spieler seiner Cooldowns.

Grundsätzlich konnte man sagen das man voran kam. Die Übersicht Stand, problematisch war jedoch das zu oft separat voneinander programmiert wurde und Variablen und Funktionen, die eigentlich genormt werden konnten, nicht einheitlich bestimmt wurden. Später wurde das öfters zum Problem.

## **Bibliotheken**

Nach dem gescheitertem Versuch JavaFX zu benutzten, entschieden wir uns dagegen Aufgrund von 2 Gründen mit Java.Swing und teilen von java.awt zu Arbeiten. A) ist es standardmäßig im Programm vorhanden und braucht lediglich einen Import, somit ist es einfach an den Schulcomputern zu implementieren und B) gibt es mehr Möglichkeiten leichter und schneller die Bibliothek zu lernen durch ein besseres Angebot an Tutorials und Erklärungen.

**Importierte Bibliotheken:**

* Javax.swing.\*
* Java.awt.\*
* Java.awt.event.\*
* Java.util.\*
* Java.io.\*
* Java.net.URL
* Javax.sound.sampled.\*
* Javax.imageio.ImageIO

# **Programm**

### **Klasse**

### ***Main(gemeinsam):*** Main ist das Haupt Programm, wo das Spiel gestartet wird. Es wird eine Instanz der Klasse StartBildschirm erstellt, die den Startbildschirm des Spiels erzeugt. Dannach wird eine Test Nachricht in der Konsole ausgegeben, um zu überprüfen, ob es keine Probleme gibt. Zusätzlich wird noch eine Instanz des SpielPanel erstellt und die Musik wird mit dem Befehl playMusic gestartet.

***StartBildschirm(Leon):***

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungDie Swing Utilities Methode wird verwendet, um sicherzustellen, dass bestimmte GUI-Updates oder Operationen im Event Dispatch Thread (EDT) durchgeführt werden, der der Hauptthread ist, der für die Verarbeitung der verantwortlich ist. Der Hauptthread wird in SpielPanel verwendet. Es werden Frames für das Spiel, den Settings und die Credits erstellt sowie String ein mit den Sprachen Deutsch, Französisch und Englisch, die man dann mit der Combobox in den Einstellungen (Settings) anpassen kann. Mit startBtn kann das Spiel gestartet werden. Die Abfrage actionPerformed überprüft ständig ob der Button gedrückt wird. Wird der Button gedrückt,wird durch dispose der frame des Startbildschirm geschlossen und das Spiel wird geöffnet.

In den Credits wurde ein Label erstellt, dass einen Text ausgibt. Da der Zeilenumbruch mit /n nicht so funktionierte wie wir es wollten, haben wir es mit <html> und dem Befehl <br> geschafft einen Zeilenumbruch zu erzeugen. Mit setForeground kann man die Schriftfarbe anpassen.

**SpielFrame(Leon):** Der Spielframe wird erstellt, sobald der Start Button in StartBilschirm gedrückt wird. Dieser Erstellt sofort das Notwendige Panel/SpielPanel. Damit der Panel funktioniert bzw. angezeigt werden kann, braucht er einen Frame (in unserem Fall der SpielFrame). Bei dem befehl Pack wird der Rahmen so angepasst, dass der gesamte Inhalt die gewünschte Größe erreicht oder überschreitet.

Mit set title kann der name an dem oberen Rand eingetragen werden. Set resizable verhindert das vergrößern bzw das verkleinern des frames. Damit das Spiel sich an die Größe des Bildschirms anpasst, wird der Befehl setExtendedState verwendet und auf Maximum\_BOTH eingesetzt. Both steht dafür dass sowohl Horizontal als auch vertikal die gesamte Größe des Bildschirms verwendet wird. Zusätzlich startet das SpielFrame noch den gameThread mit der Methode startgameTread().

***SpielPanel(Gemeinsam):*** Damit der GameThread überhaupt benutzt werden kann, muss er noch Implementiert werden, dies wird mit dem Befehl **implements Runnable**  gemacht. Der Thread ruft automatisch die Methode run() auf, die eine Schleife enthält, die den Spielzustand aktualisiert und das Panel durchgehend neu zeichnet. Die Methode update wird verwendet, um den Spielzustand basierend auf Benutzereingaben (Tastatur- und Mausereignisse) zu aktualisieren. Innerhalb der Update Methode werden die Handler Klassen Mouse- und Keyhandler aufgerufen(werden später weiter erklärt). Die Methode PaintComponent() ist zuständig für das Einfügen von Grafiken und ist eine von Java verfügbare Methode. Als Parameter hat sie die Graphics. In Ihr muss eine super Methode verwendet werden. Mit 2 for-Schleifen wird das Spiel gezeichnet. Beide Schleifen setzen Rechtecke in der Größe der Cell\_size (30) bis die i die Anzahl der Field\_size (32) hat. Darin wird abgefragt ob das momentane Feld ein Randblock ist, wenn ja wird es Grau gefärbt ansonsten Weiß. Zusätzlich wird mit dem Befehl graphics.fillRect() der Rand jedes Kästchen schwarz gefärbt.

Zu Spielbeginn werden 2 zufällige Zahlen erstellt zwischen 0 und 30. Diese werden Für die Position der Münzen benötigt.

Ein Bild, das Text, Screenshot, Uhr, Schrift enthält.

Automatisch generierte Beschreibung

Sobald die Position der Münze mit der Position des Spielers übereinstimmt, wird das Feld der Münze weiß gefärbt und erneut die Methode randomNum() aufgerufen, um eine neue zufällige Position der Münze zu berechnen, sowie die Methode Countgold, die Anzahl des Gold des Spielers um 1 erhöht. Dannach wird der neue Wert der Münze zurück gegeben und mit dem alten wert Überschrieben. Durch Update() wird die Münze dann an die neue Position gesetzt.

Die run() Methode ist dazu da um zu bestimmen wie oft pro Sekunde das Spielpanel aktualisiert wird und wird um Thread ausgeführt. Die Zeit wird in nanosekunden gemessen und muss in long angegeben werden.

In der von Java vorgegebene Methode update() werden die von dem MouseHandler und vom KeyHandler definierten Inputs in Befehle umgewandelt. Die ersten 4 ifs behandeln die W/A/S/D Tasten die zur Steuerung des Spielers dienen. Wird eine (oder auch mehrere) betätigt der Spieler um den Wert der Zelle nach oben/links/unten/rechts bewegt.

Ein Try Befehl gibt der Bewegung einen Delay bei Möglichkeit, sollte aus irgend einem Grund das nicht funktionieren gibt Catch eine Fehlermeldung aus.

Zusätzlich werden die Fähigkeiten des Spielers, also die jeweiligen Methoden castSwordHit() und castShieldBlock() bei Betätigung des Links- oder Rechtsclicks ausgeführt.

***Charakter(Marc):***

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungDie Charakter Klasse erbt aus der Entity seine primitivsten Daten und Methoden. Die Charakter Klasse selbst definiert die Aktion und die Schadensberechnung des Spielers. Für jede Fähigkeit wird ein Timer (Cooldown) definiert der Startet sobald eine Fähigkeit ausgeführt wird. Während der Timer läuft ist es nicht möglich die jeweilige Aktion erneut auszuführen. Stop und Start des Timers werden durch startCooldown() und stopCooldown() Methoden abgefragt. ActionPerformed() überprüft ob ein Timer abgelaufen ist.

CastSwordHit() ist die Methode die einen AoE (Area of Effect) Angriff startet. Bei der Benutzung werden alle Felder im 2 Zellen Radius des Spielers in zwei große double Arrays eingeteilt. swordAoE\_X und swordAoE\_Y bekommen dann die Koordinaten der Zellen durch eine Rechnung eingeteilt (z.B. swordAoE\_X [0] [0] = SpielPanel.playerX - 2\*SpielPanel.CELL\_SIZE;). Dann wird abgefragt ob der Cooldown läuft oder das Schild aktiv ist, ist dem nicht der Fall startet die Aktion selbst und der Cooldown wird gestartet. Ansonsten passiert nichts. Eine doppelte For-Schleife überprüft dann die Koordinaten der zwei double Arrays mit den Koordinaten des Mobs. Stimmen diese überein wird die Variable swordHitConnects auf true gesetzt und nimmt Schaden. Ansonsten passiert auch hier nichts. Nach dem ablaufen des Cooldowns kann der Vorgang wiederholt werden.

Angedacht war auch die CastShieldBlock() Methode. Sie sollte eine Fähigkeit werden die Mobs aufhält schaden am Spieler zumachen . 5 Sekunden lang kann der Spieler sich bewegen ohne Schaden nehmen zu können. Währendessen kann er selbst keinen Angriff ausüben.

Am Beginn des Codes wird wieder ein Cooldown (cooldownShieldBlock) abgefragt, ist dieser hier nicht Aktiv (10 Sekunden) startet die Aktion. Es wird zusätzlich zum Cooldown der Methode ein Timer aktiv (activeTimerShield) der 5 Sekunden lang das eigentlich Schild vor schaden aktiviert.

Am Ende des Codes der Charakter Klasse sind noch zwei Methoden. Die reduceHealth() Methode senkt das Leben des Spielers um den Faktor 1 und regenerateHealth() sollte beim Wechseln des Raums das Leben des Spielers auf den Standardwert 3 zurücksetzten.

***Mobs/HandlerCreature(Jason):*** Für die Gegner haben wir die Klasse Mobs und HandlerCreature. In der Handler Klasse wird eine Arraylist erstellt um Objekte zu speichern die in der Mob Klasse entstehen. Die Methode Tick() aktualisiert das vorhandene Objekt. Bei Render() wird die Grafik also das Bild wiederholend rangeholt. Die Mob Klasse erbt von der Entity Klasse die X und Y Koordinaten und ID. In der tick() Methode wird die Schnelligkeit festgelegt (Durch velX und velY) dazu werden Rechnungen ausgeführt für das Verfolgen(Distanz) von dem Spieler. Nach get Methoden kommt das Rendern von der Grafik des Mobs. Es war noch vorgesehen Animationen für die Gegner zu machen, deswegen habe ich eine Arrayliste für die Grafiken gemacht, doch das war zeitlich nicht möglich.

***SpriteSheet(Jason):*** Durch die Spritesheet Klasse kann ein Bild (SpriteSheet) abgerufen werden.

***Sound(Marc):***

Die Sound Klasse ist eine direkte implementierung des Codes für Sound und Musik aus dem YT-Tutorial: "Sound - How to Make a 2D Game in Java #9" von Ryi Snow (Siehe Quellenangaben). In dem Code wird ein URL Liste als Array definiert wodurch Sound Dateien (z. B. .wav oder .mp3) verwaltet werden.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte BeschreibungAm Anfang wird im Try-Block versucht, eine Audio-Datei zu öffnen. Dafür wird die Methode getAudioInputStream() aufgerufen und der entsprechende Dateipfad übergeben. Der Index "i" wird verwendet um die richtige Audio-Datei auszuwählen. Der jeweilige Sound oder die Musik wird in audioInputStream gespeichert.

Anschließend wird ein neues Audio-Objekt der Klasse Clip erstellt und in als clip gespeichert. Die Methode open() wird aufgerufen und der zuvor geöffnete audioInputStream() als Parameter übergeben. Dadurch wird der Audioclip mit den Daten aus der Audio-Datei gefüllt.

Sollte es ein Fehler geben wird dieser einfach ignoriert und es spielt keine Musik ab.

playSound() startet dann die Audiodatei, loopSound() spielt diese nicht nur ab sondern wiederholt auch diese immer wieder (Dauerschleife) und stopSound() beendet eine Sounddatei während der Benutzung.

***Handler(Jason, marc):***

Die Key-Mouse Handler Klasse nimmt die Eingabe von der gedrückten Taste und speichert diese in eine Variable die dann abgefragt wird ob diese mit der gewollten Taste übereinstimmig ist. Wenn es übereinstimmt, wird die Boolean variable true und beim los lassen false gesetzt. Die Leeren methoden in der MouseHandler Klasse werden benötigt um die Maus zu bewegen.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Schrift, Screenshot, Quittung enthält.

Automatisch generierte Beschreibung

**Entity(Jason):** Der Konstruktor gibt die Position, die Art(ID) und das Leben an. Sie können durch Get Methoden aufgerufen werden. Die Charakter Klasse vererbt Entity genauso wie Mobs.



***ID(Jason):*** Die ID-Klasse ist keine normale Klasse, sie wurde durch Enum zur Liste gemacht. Enum steht kurz für Enumeration, was so viel heißt wie Aufzählung.

***BufferedImageLoader(Jason):*** Der BufferedImageLoader benutzt die Bibliothek ImageIO um ein Bild wieder zu geben.

## **Ressourcen**

### **Sprites**

Der Spieler und der Mob wurde von Leon selbst gepixelt.

### **Sound**

Als kleinen Bonus hat Marc zusätzlich zum Spiel selbst noch einen kleinen, simplen Soundtrack mit dem Programm FL Studios geschrieben. Der Sound wird über die Main Klasse aufgerufen und in der Sound Klasse gespeichert

# **Fazit**

Wie man sieht, waren waren wir etwas zu ambitioniert und wurden von einigen Problemen geplagt, so dass wir schon nach einigen Tagen gemerkt haben, dass einige Ziele unrealistisch sind. Würden wir wieder ein Projekt starten, würden wir definitiv mehr Zeit in die Vorbereitung und die Ideen an sich Investieren, da dies uns echt viel Zeit im Programmiert Part gekostet hat. Das Endprodukt ist lange nicht fertig, und es kann noch vieles Verbessert werden, allerdings sind wir für unser erstes Projekt mit vielem zufrieden.

# **Quellenangaben**

<https://youtu.be/om59cwR7psI>

<https://youtu.be/VpH33Uw-_0E>

<https://youtu.be/oPzPpUcDiYY>

<https://youtu.be/6Tj6XYGWfko>

<https://youtu.be/Kmgo00avvEw?t=8296>

<https://youtu.be/nUHh_J2Acy8>

<https://www.youtube.com/watch?v=nUHh_J2Acy8>

<https://www.youtube.com/@thesimpleprogrammer3099>